THE BELL SYSTEM TECHNICAL JOURNAL
Vol. 62, No. 1, January 1983
Printed in U.S.A.

The 3B20D Processor & DMERT Operating System:

Field Administration Subsystems

By R. H. YACOBELLIS, J. H. MILLER, B. G. NIEDFELDT, and S. S. WEBER

(Manuscript received March 10, 1982)

This article describes the field administration facilities of the Duplex Multiple Environment Real Time (DMERT) operating system, as provided on the 3B20D Processor. These facilities are: Recent Change/Verify, the subsystem that allows manipulation of office-dependent configuration information; Field Update, the software change mechanism; and System Update, the component used to install a new generic program in an office. The article also includes information on how these capabilities fit into the overall scheme of field support in an in-service office environment.

I. INTRODUCTION

An integral part of high-reliability applications of the Duplex Multiple Environment Real Time (DMERT) operating system is the administration of system hardware information and of software. This includes both the initial delivery of the system as well as subsequent upgrades. In DMERT[1] there are three commonly used capabilities to apply, track, and administer such changes. These are Recent Change/Verify, Field Update, and System Update. They are listed in this order according to decreasing frequency of field use and increasing impact (typically) on the overall system. Each of these capabilities is designed to permit display of some aspect of the current status of the system, to change that status in a simplified and highly reliable way, and to either reverse such changes or make them permanently a part of the system. This article discusses each in turn, and provides examples of their use. Each capability may form the base for an application-dependent version of its function. These functions are discussed briefly in the rest of this introduction.

The 3B20D Recent Change/Verify (RC/V) system provides the ability to change and manipulate various aspects of office-dependent information. This capability is focused on the system hardware and software configuration and is based on the Low-Level Access (LLA) Data Base System, whose operation is normally hidden from field-site administrators. RC/V is used manually or automatically to verify and change the hardware and software components known to the system, and the ways in which they are interconnected.

Field Update is used to correct problems in the operation or functionality of the system. Field Update is the official fix mechanism for DMERT. Rapidly installed emergency fixes, as well as more routine trouble corrections, may be installed into the software or other files in DMERT via Field Update.

Finally, System Update, also known as Generic Update, changes a major portion of the entire DMERT or application generic program. In doing so, System Update may write over old generic information or provide a completely restructured generic program image. Typically, a new generic release will involve a new structure for RC/V information as well, so RC/V may be involved with such an update. The following sections provide more details on these fundamental administrative capabilities of DMERT.

II. RECENT CHANGE/VERIFY—LOW-LEVEL ACCESS DATA BASE SYSTEMS

The 3B20D/DMERT System has provided a data base management capability as part of the DMERT operating system. Built upon a Low Level Access (LLA) data base system are the Equipment Configuration Data Base (ECD), System Generation Data Base (SG), and the 3B Recent Change/Verify (RC/V) and Data Base Evolution Systems. This section describes these systems and their relationship to the field administration environment.

2.1 Low-Level Access Data Base System

The Low-Level Access Data Base System organizes and manipulates data in a C-language environment. The name low level implies that the system places minimal restrictions on its users: decisions about data organization and retrieval are left to the application. LLA trades user convenience for greater flexibility in data base design and performance tuning.

LLA gives the user latitude in defining both data units and data models and provides a powerful set of primitives to access the data. System characteristics include:

(i) Data definition via a hierarchy of abstract types

(ii) Specification of data mapping from the data base to the user's buffers

(iii) Ability to select various access methods, i.e., logical organization of subsets of data

(iv) Data access through a library of functions

(v) Isolation of operating system dependencies in a small number of program modules.

Figure 1 gives a simplified schematic of the operation of an LLA application.

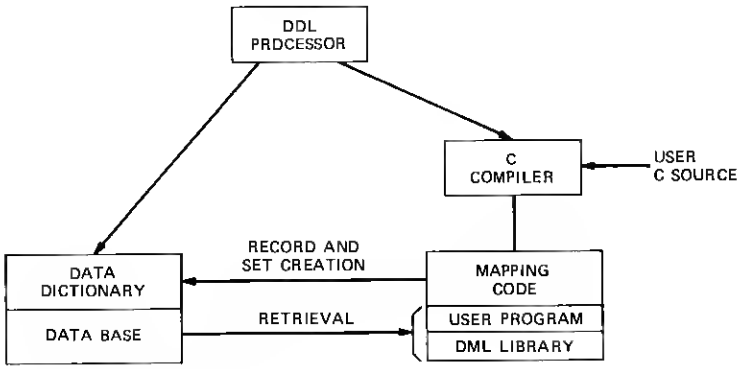
2.1.1 Data definition

The Data Definition Language (DDL) is used to define the "shapes" of records, the LLA data type for retrieval and storage. It also allows user-defined "views" of the data base via data mapping, and the specification of data models by associating records with access methods. The recognizer for the DDL, the Data Definition Language Processor (DDLP), has many C-compatible features, such as common syntax for preprocessor lines, comments, identifiers, constants, and type definitions. The DDLP generates C code to implement data mapping and C definitions, and a data dictionary to describe data types.

2.1.2 Data manipulation

The Data Manipulation Language (DML) is a library of functions that perform actions on instances of the data types defined by the DDL. The DML provides the following facilities:

(i) Creation and deletion of instances of data types



DDL – DATA DEFINITION LANGUAGE
DML – DATA MANIPULATION LANGUAGE

Fig. 1—Low-level access application.

(ii) Retrieval and update of existing instances of data types
(iii) Gathering of information about existing data instances.

These categories exist for instances of data bases, sets, and records. Generally, the lifetime of an instance of a data type starts with creation, proceeds through several retrievals and updates, and ends with deletion.

LLA is not used directly by a field administrator. Instead, the creators of various LLA data bases, be they 3B20D/DMERT system programmers or 3B20D application designers, provide appropriate higher-level access to their particular LLA data base application.

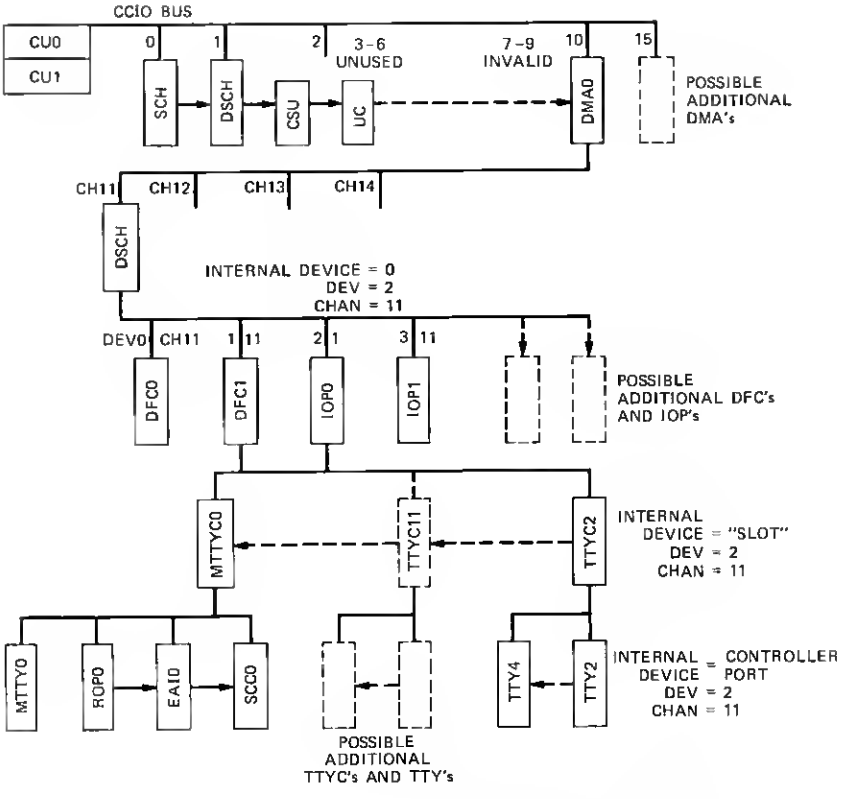
2.2 3B20D Data Base Recent Change and Data Base Evolution Systems

2.2.1 3B20D data bases

The 3B20D/DMERT operating system has two major LLA databases. The Equipment Configuration Data Base (ECD) describes the processor and peripheral hardware configuration, while the System Generation (SG) Data Base describes the system parameters, boot processes, and disk image and ECD administration information. The concept of a data base was adopted to eliminate redundant device information, provide a unified approach to handling and accessing that information, and provide easy methods for generating and changing it.

Records in the ECD data base represent the hardware devices in the 3B20D Processor system, such as the Control Unit (CU) and Input/Output Processor (IOP), and are logically linked in a manner analogous to the physical linkages (see Fig. 2). In addition, records are provided to organize physical devices as logical devices and to maintain error counts for each physical device. To provide rapid access, the ECD is always kept in main memory.

The information in the ECD and SG data bases is used by several classes of users. The DMERT operating system, itself, forms one set of using processes and includes the device drivers, processor and peripheral diagnostics, and processor and peripheral fault-recovery programs. The second class of users of these data bases is the human user, whether that person be a Bell Laboratories' application designer adding new peripherals to the ECD or an operating company craft preparing to add more memory to an on-line 3B20D in the field. Two types of access have been provided for these two classes of users: The DMERT operating system processes access the ECD through a collection of LLA primitives that provide rapid access to those specific items required, for example, by the device drivers. Human users utilize the Recent Change/Verify system, which provides a forms-oriented input, via a cathode ray tube (CRT) terminal. The user may create, change, delete, or merely review the forms. Error and consistency checking is provided at the time of initial entry and before storage into the data base.



CCIO – CENTRAL CONTROL I/O BUS
CH – CHANNEL
CSU – CACHE STORE UNIT
CU – CONTROL UNIT
DEV – DEVICE
DFC – DISK FILE CONTROLLER
DMA – DIRECT MEMORY ACCESS
DSCH – DUAL SERIAL CHANNEL
EAI – EMERGENCY ACTION INTERFACE
IOP – INPUT/OUTPUT PROCESSOR
MTTY – MAINTENANCE TERMINAL
MTTYC – MAINTENANCE TERMINAL CONTROLLER
ROP – RECEIVE ONLY PRINTER
SCC – SWITCHING CONTROL CENTER
SCH – SERIAL CHANNEL
TTY – TERMINAL
TTYC – TERMINAL CONTROLLER
UC – UTILITY CIRCUIT

Fig. 2—Prototype 3B20D configuration.

2.2.2 *3B20D Recent Change/Verify*

The Recent Change/Verify system is built upon the LLA data base management system and utilizes the LLA primitives for accessing and managing its two DMERT data bases. There are three basic components of 3B20D RC/V (see Fig. 3). The first is the front-end form processing system. This component is known as the On-line Data Integrity (ODIN*) subsystem. ODIN allows the various forms to be specified through a series of CRT screen mask definitions and for each

* ODIN is a product of Western Electric Company.

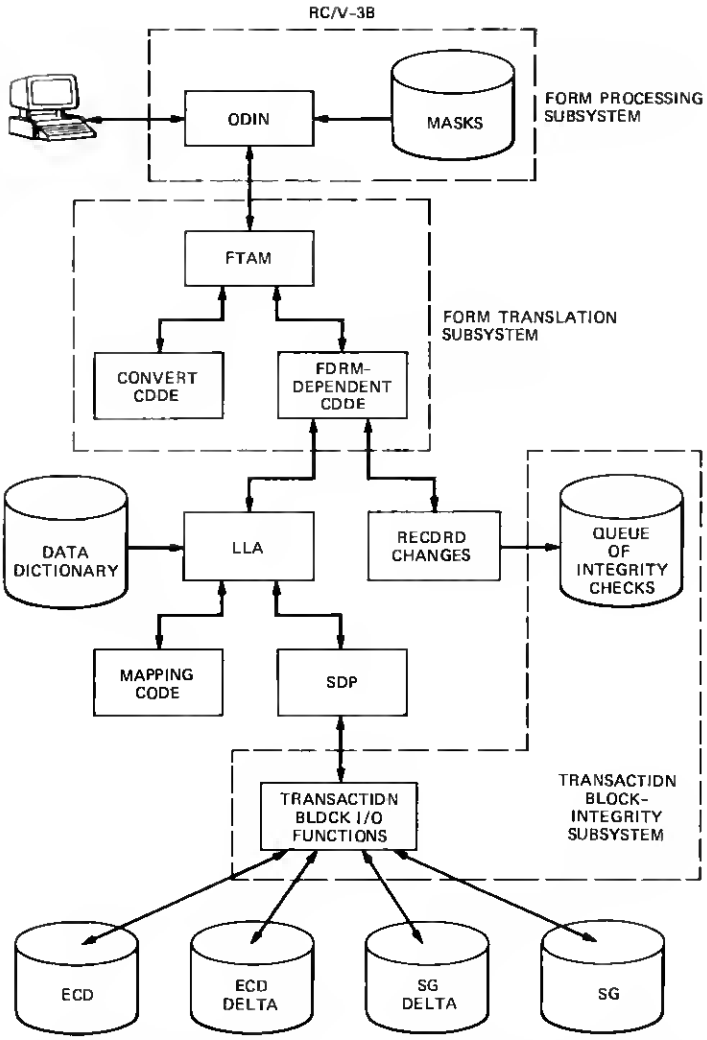


Fig. 3—Components of Recent Change/Verify.

of these definitions to contain certain syntactic information to be checked upon entry. For the ECD/SG data bases there are 36 different form types, each of which has an associated mask definition. Most forms are either ECD or SG forms, but there are a few that are directives for the RC/V or Evolution systems. For each form type some error checking is provided. The second fundamental component

of RC/V is the Form Translation and Mapping subsystem. This takes the output of ODIN and transforms it into LLA record definitions and access functions. Then the LLA functions are used to actually manipulate the data in the ECD and SG data bases. The third component is the transaction block-integrity check subsystem. This provides a mechanism for checking consistency between forms. RC/V has implemented the concept of a "transaction." Two special forms delimit a transaction. Upon processing a transaction-end form, RC/V invokes the integrity checks as well as linking the new information into the data base.

As we stated earlier, the ECD that describes the running 3B20D is always in main memory; however, there is also a copy on the disk. In order for a change to be made permanent it must be applied to the disk as well as the memory version. To maintain the integrity of the ECD, changes are soaked on the memory version (test state) before they are applied to the disk version (active). A special form has been provided to perform this final step of activating changes to the disk copy of the data base. Upon processing of this form, RC/V copies the main memory copy of the ECD to the disk. To facilitate error checking and correction, a journal file of all transactions is kept on-line and can be printed on the Receive-Only Printer (ROP) at the request of the office craft. Also, an error log file is maintained and a periodic audit of the ECD structures is performed.

2.2.3 Data Base Evolution System

Because the release of a new 3B20D/DMERT generic is anticipated to be associated with changes to the ECD or SG forms or the LLA primitives, a system for transforming these data bases has been provided. The Data Base Evolution system (DBEVOL) allows this transformation to occur in a regular and uniform manner without special programs needing to be written. DBEVOL allows old data to be restructured, new data fields to be added to existing forms, and old data to be deleted or changed. DBEVOL also provides semantic hook functions that allow applications to tailor some specific information before completing the data base evolution.

DBEVOL has two types of steps. The first set is characterized as pre-processing. Here a translation data base (also an LLA data base) is built on a host support processor. The inputs are the old and new form specifications (as used by RC/V) and a specification of the changes in Form Translation Language. These inputs are supplied with the new DMERT generic program. If semantic hook functions are required by the application they are also an input to the final translation data base. A translation data base matching the required changes in the standard DMERT ECD is also released with new DMERT generics.

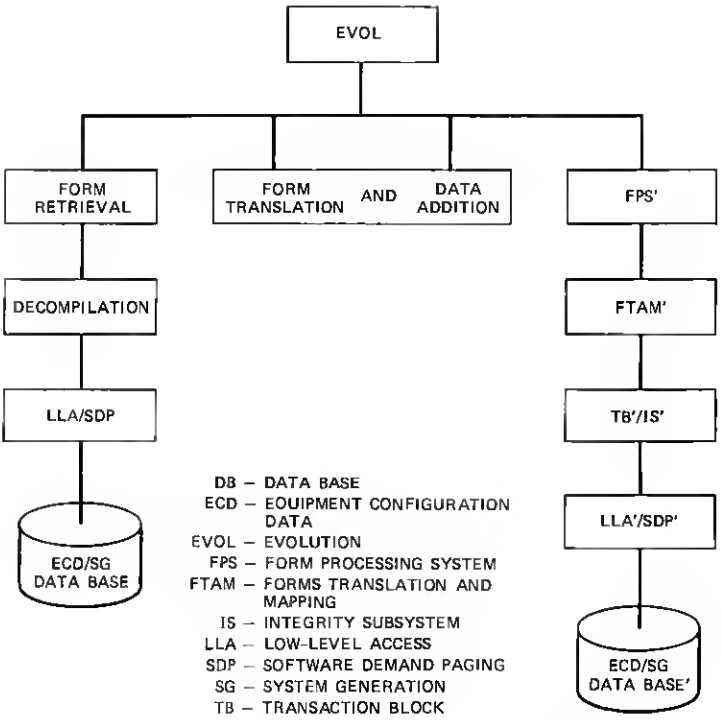


Fig. 4—Evolution of 3B20D/DMERT Data Base Management System.

The second set of actions are run-time steps that produce a new, evolved ECD/SG data base pair (see Fig. 4). The first step is a dump of the old ECD using the "old" existing generic RC/V. This is produced using one of the special forms provided by the RC/V system. Then this snapshot of the old data base is translated into a snapshot of the new data base. The "new" RC/V is then used to load the new data bases into the proper LLA format for the 3B20D.

DBEVOL runs on both the support processor and the 3B20D giving the using applications considerable flexibility in choosing a strategy for performing data base evolution. The evolved data base is actually put in place on the running 3B20D during the generic update scenario described below.

III. FIELD UPDATE

Field Update, which is typically called "overwriting" in traditional Electronic Switching Systems (ESSs), is the problem correction mechanism for DMERT. While overwriting usually applies specifically to program bugs, Field Update may be used to correct any file on the 3B20D disk. Such files may contain human-readable text or binary

tables, for example. (In DMERT, files are structured like a *UNIX** operating system file system.[2]) Field Update must perform this updating without disturbing call processing or other critical system functions. Since operating systems do not normally support this style of updating, some difficult technological problems had to be overcome in designing and implementing Field Update. Some of these problems and their solutions are described below, followed by a more general discussion of the overall structure and use of Field Update.

3.1 Problems and solutions

Like most modern operating systems, DMERT supports the concept of a process, which is a collection of tightly coupled executable programs. Programs are in turn broken down into units that perform specific activities, called functions. Processes can communicate with each other, generally at "arms-length," and are normally protected from each other by DMERT software and the 3B20D hardware and microcode. Since Field Update runs as a cooperating set of processes within DMERT, some highly specialized operating system interfaces were required to break through this protection. Furthermore, the real-time critical processes in DMERT or its applications must run continuously [they are termed "non-killable" (NK)], so that they are always available to process events quickly. The running process images of such processes must be accessible and changeable in main memory, again via special operating system functions.

Since a process is a collection of functions, the C-language[3] function was chosen as the unit of update. The implementation of field update specified that there be a single reference point for each changed function, so as not to require changes everywhere such a function was involved. To solve this, the concept of a Transfer Vector (TV) used in ESSs was implemented within a process image. Figure 5 is an example of a simplified process image showing this. In Fig. 5, the TV area contains a list of the addresses of the process's functions. When a change is made to function f, the new version f′ is written into a special "patch" area provided with the process, and the particular address in the TV area is switched to point to f′ (see Fig. 6). This solution also allows the fix to be backed out by changing the address in the TV back to its original value. When the fix has been tested and is ready to apply permanently, the space occupied by f can be made available for future fixes (Fig. 7). While this concept is simple, introducing TVs to DMERT had operating system implications down to the microcode level. With TVs, the impact of introducing a new or changed function

* Trademark of Bell Laboratories.

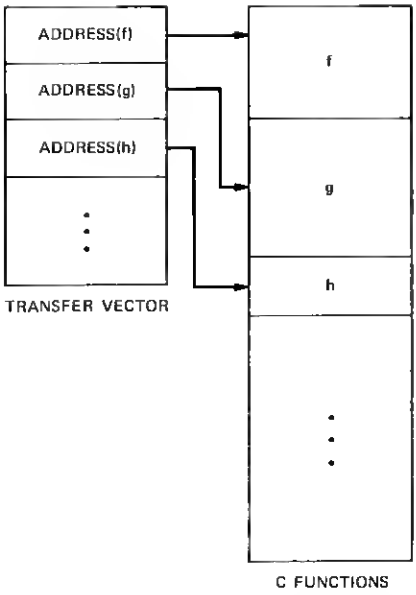


Fig. 5—Simplified DMERT process image.

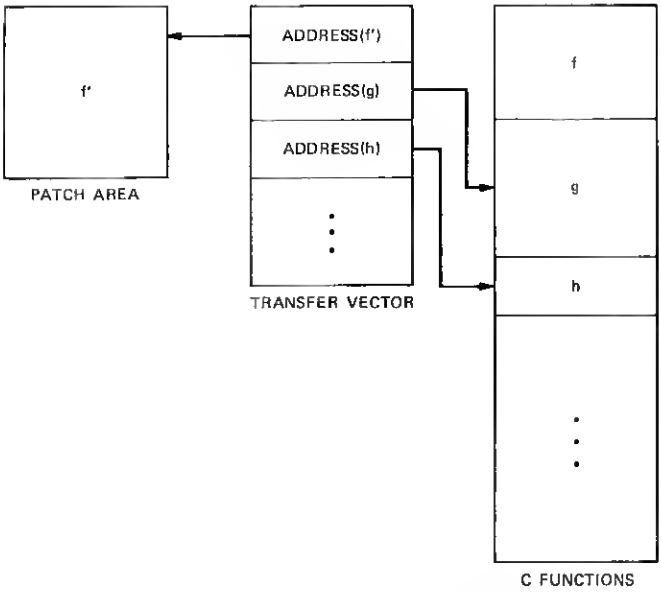


Fig. 6—Function f replaced by function f'.

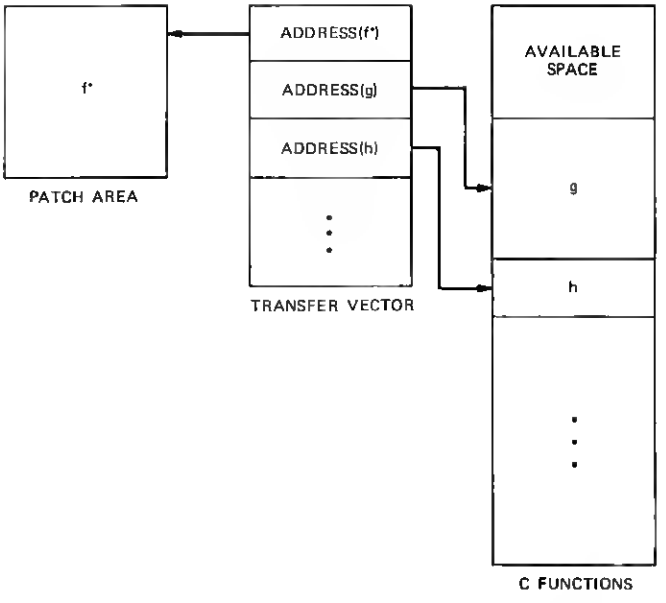


Fig. 7—Reclaiming the space occupied by function f.

has been restricted to a small, well-defined area of the process, making this activity inherently more reliable.

Traditional operating systems do not have the ability to change a critical function or process while the system is running. Since DMERT is derived from such an operating system, many challenges were encountered in providing the field update capability. Some specific areas included:

(i) The ability to change a file both instantaneously and in a temporary way. This is used in updating both non-killable processes and more routine processes that can be terminated and restarted;

(ii) Retention of sufficient symbolic information to properly update the 3B20D disk-resident versions of processes ("pfiles");

(iii) The ability to update C functions even though the old versions of the functions had been suspended while field update was running;

(iv) The ability to change data contents or the structure of data used by a continually running process;

(v) The ability to coordinate changes to functions within a process.

3.2 The use of field update

Field Update is an end-to-end concept within DMERT; that is, it is involved with the development, distribution, installation, and tracking of changes. When a process is first introduced into DMERT, or when its subsystem architecture changes, the process developer must com-

municate its characteristics to personnel who administer the DMERT source programs. The developer also must create a script of commands to be executed at a field site, which will be used to install, back out of, or make permanent a fix to the process. Generally, this will be simple to do because there are categories of existing process scripts, and new processes will fit into an existing category (or a simple modification to one will suffice). Once these steps are taken, the developer can depend upon the DMERT administrative system[4] and specific Field Update change development commands to remember these details. This approach standardizes the development of fixes so that each is handled the same way, as opposed to being a unique activity. The primary advantage comes when an emergency fix must be created quickly without the extra burden of collecting procedural information.

When a developer has created a fix and tested it, the standard change development mechanisms produce a package called a Broadcast Warning Message (BWM), which is used to transmit and install the fix (see Fig. 8). System Test personnel use this package to test the

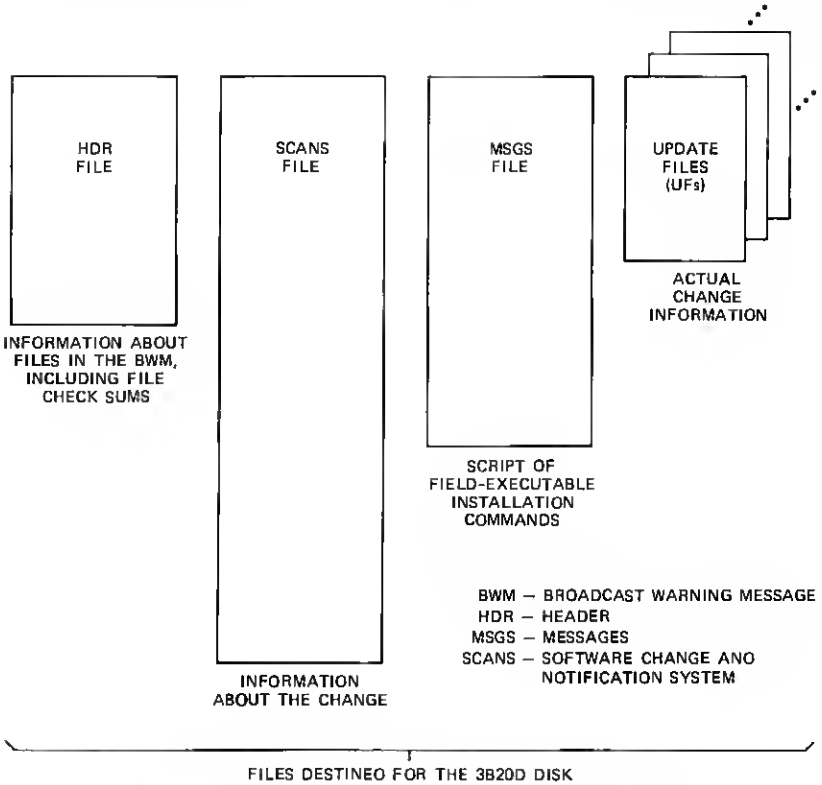


Fig. 8—Structure of a broadcast warning message.

field updatability of the fix as well as its impact on the system in the same way it will be installed at a field site (see the article on "System Integration and Test" in this issue of the Journal). When testing is completed, the fix can be packaged together with other fixes via automated tools into an official BWM for delivery to application project personnel, who will intermix it with application-specific BWMs and send it on. During this packaging, the particular order of installation of specific fixes is indicated both within and across BWMs.

A BWM consists of a set of files in a *UNIX* operating system directory, and can be transmitted via magnetic tape to a site. The Bell System is standardizing on the Software Change and Notification System (SCANS-II) as the official change distribution network, and the files in a DMERT BWM are also compatible with SCANS-II. DMERT also provides file reception software for use with SCANS-II. Typically, personnel at a Switching Control Center (SCC) will interrogate SCANS-II, recognize that a change is pending for one of their associated field sites, and initiate transmission of the change to the field site.

Once a change reaches a field site, it is stored in a staging area on disk until it is manually installed. The developer-produced script of commands is sent as part of the BWM (see Fig. 8), and is used by office personnel to install the change. With a short sequence of DMERT Field Update commands, the fixes can be:

(i) Installed
(ii) Tested
(iii) Backed out or made permanently a part of the system.

While a fix is being installed, an internal system error will result in automatically backing it out; once it is soaking in a temporary state, it may be backed out manually, or automatically if the system undergoes a major recovery action.

Each field site maintains an on-line log of all Field Update activity since the last System Update (see Section IV). This may be used to verify the current state of the office as far as installed BWMs are concerned, and is used each time a new change is installed to guarantee proper sequencing of changes. Other Field Update-related utility programs in DMERT can be used to print out a C function-to-process address map, and to verify that the main memory (executing) copy of a process matches its image on the 3B20D disk (see Section 3.1).

By the facilities mentioned above, Field Update allows fix creation in a style compatible with normal program development, prepackaging of developer-approved installation scripts, fix coordination both within and across BWMs, automated delivery and installation mechanisms, and detailed change tracking. These capabilities make Field Update a truly end-to-end DMERT change mechanism.

3.3 Field update example

Let us presume for this example that a problem has been found in the DMERT disk driver program, whose pfile is called dkdrv.o in directory /bootfiles. The developer has constructed a fix and tested it, and further system impact testing has verified it. The fix is given a DMERT official BWM name of BWM82-0028 (the first two digits are the year, and the last four a sequence number), and is passed to personnel in an application of DMERT, who approve it and send it out as application BWM, BWM82-0037. Once the fix has arrived at a field site, it is installed via the commands shown in Fig. 9. The descriptions below explain the commands:

(i) Request a printout of change information that field update has logged against process dkdrv.o.

(ii) Prepare the site to receive the BWM. After SCANS-II receives a command to send the BWM (not shown), it is transmitted automatically to the site with data error detection and positive reporting.

(iii) Install the fix into the system.

(iv) Test the fix (coupled, perhaps, with manual actions).

(v) Make the change permanent and remove the BWM files from the system. In this particular case the DMERT boot image is rebuilt as part of making the fix permanent, because the changed process is one of the system boot processes.

(vi) Once again display the change status of dkdrv.o.

(vii) Print a map of C functions and their addresses for drdrv.o.

(viii) Reclaim the space occupied by old versions of C functions in dkdrv.o.

The installation command mentioned above causes an entire set of commands to be executed, those in the "install" section of the script originally provided by the developer. An example of that script is shown in Fig. 10, which shows the Messages (MSGS) file for BWM 82-0037.

```
(i)    UPD:DISPLAY; FN "/bootfiles/dkdrv.o"!
(ii)   IN:REMOTE:START!
       VFY:BWM: 82-DD37!
(iii)  UPD:BWMNO 82-0037!
       UPD:EXEC 82-DD37: CMD APPLY!
(iv)   UPD:EXEC 82-0037; CMD SOAK!
(v)    UPD:EXEC 82-DD37; CMD OFFICIAL!
       CLR:BWM:ALL!
(vi)   UPD:DISPLAY; FN "/bootfiles/dkdrv.o"!
(vii)  UPD:TRC; FN "/bootfiles/dkdrv.o" : ALL!
(viii) UPD:AUD!
```

Fig. 9—Commands to Receive and Incorporate BWM 82-0037.

```
  APPLY.
                MRs: d8200002; DMERT BWM82-0028
UPD:UPNM BWM82-0037;FN"/boottiles/dkdrv.o":UF"/etc/bwm/82-0037/one.m"!
  SOAK.
                The tix(es) should soak for at least 1 days 00 hours 00 minutes.
                It will be apparent that the tix(es) have been applied:
                When no disk restore failures occur,
                commands to soak the fix appear here.
  BKOUT.
                If the tix results in the need to reboot the system, the fix will
                have been backed out automatically. If the tix does not result
                in a reboot but otherwise does not work correctly, it can be backed
                out by entering the command [s]:
UPD:BKOUT;UPNM BWM82-0037!
  OFFICIAL.
UPD:UPNM BWM82-0037;OFC!
                This will update the boottile APPDMRT.
```

Fig. 10—MSGS file for BWM 82-0037.

IV. SYSTEM UPDATE

DMERT System Update provides a safe, reliable mechanism for field personnel to introduce new versions of DMERT and application software into 3B20D/DMERT systems, while minimizing service disruption. System Update differs from Field Update in the magnitude of the program and data changes being installed. Normally, a system update will replace all the software in the system with the release of a new generic program, which is a complete reissue of DMERT and/or application software and/or data. For this reason, system updates always include a memory reinitialization with a full bootstrap (reinitialization of all processes and data from disk). Only the contents of protected application segments, special memory areas where application systems may retain critical information, are retained across the boot. Since a system update includes a reinitialization, only the version of the software on the 3B20D disk is updated. The main memory images of system processes will then be re-read from the disk during the bootstrap. This section describes how this disk updating is done within DMERT, and gives an overview of the overall System Update process.

4.1 System Update concepts

The DMERT System Update Program (SUPR) provides a way to replace the entire contents of the 3B20D disk with a new version of those contents from a magnetic tape. SUPR deals with masses of data, and changes the disk contents section by section rather than file by file or logical data base updates. These sections are called *partitions*. To do this, SUPR takes advantage of the fact that the 3B20D disks are duplexed for reliability, writing the new system information onto

only one of a pair of disks. This is the *off-line disk method* of system updating. It derives its name from the fact that one of a pair of disks must first be removed from active service (taken off-line) before writing the new system onto it. With the off-line disk method the amount of redundant disk information is kept to a minimum during the update, and the disk structure may be completely changed. There is some increase in system vulnerability during the time that the disks are not running in duplex mode.

Certain aspects of the system update procedure have caused unique requirements and changes within DMERT. The key to the off-line disk method is protecting both generic programs from being overwritten during the update procedure. Since these generics reside on duplex disk mates, an off-line disk must never be restored to service. (The restore process includes a copy from the on-line to off-line disk.) The attributes of the "off-line" device state in the ECD were expanded to provide this capability. After a bootstrap on a new generic disk image, the disk copy of the old generic must similarly be marked off-line, and hence protected from restorals. This was accomplished by having each generic's ECD record the disks containing the *other* generic as off-line.

It was also necessary to be able to access partitions on an off-line disk, in order to read or write partitions on an off-line disk, to transfer files from the old generic to the new generic, and to perform recent changes on the new generic ECD (for example, in marking old generic disks as off-line). This was done by having the disk driver program access the Volume Table of Contents (VTOC)—the directory of the disk's contents—on the off-line disk during the update process. This is a special case, since the VTOC on an off-line disk may be different from that of its mate disk, or may not even be sane. When updating multiple disks, SUPR uses a special disk identifier added to the VTOC to ensure that the disk image being written corresponds to the information on that disk. As another safeguard, System Update uses checksums (special numbers computed from the data in a file) on the generic tape to check the new generic data for damage before writing it to the disk.

4.2 System update scenario

SUPR provides a complete update scenario, including a means to reverse the update and re-establish the original system. Because of the major impact on the application during a system update, the complete update procedure is broken down into several distinct steps, and allows the craft to choose the best time to begin each successive step of the update. The update may be canceled at any step of the procedure. Application-dependent processing may be introduced at any step.

Under favorable conditions only the forward steps of SUPR would be used, resulting in a successful update. These steps are:

(i) Enter new generic—Read all the new generic data onto the off-line system disk.

(ii) Proceed with new generic—Make final preparations prior to booting the system from the new generic.

(iii) Boot from new generic—Manually boot the system using the new generic.

(iv) Commit to new generic—Complete propagation of the new generic into the system after the soak period by removing all aspects of the old generic.

If the new generic does not work as expected, the craft would not commit to it, but would start a backout procedure to return to the original system.

SUPR also provides a convenient mechanism to allow application-dependent processing at each step of the update procedure. This is accomplished by transferring control to an application process that can perform whatever actions are appropriate. The types of actions most likely to be done as part of the application processing would be to transfer data (files, data bases, office-dependent information) from the old generic to the new generic or to save call registers and billing information in protected application segments prior to suspending call processing and booting from the new generic.

V. SUMMARY

This article has dealt with the subsystems of DMERT that administer changes to system data. Recent Change/Verify is used to change system configuration data and its underlying data base, Field Update allows "bug fixes" and logical file changes, and System Update will install an entirely new version of the operating system. These subsystems were described and examples given of their use. In each case DMERT provides change application, testing, and rejection or acceptance capabilities in a context very similar to that of typical operating systems, but in a highly reliable way.

REFERENCES

1. M. E. Grzelakowski, J. H. Campbell, and M. R. Dubman, "The 3B20D Processor & DMERT Operating System: DMERT Operating System," B.S.T.J., this issue.
2. D. M. Richie and K. Thompson, "The UNIX Time-Sharing System," B.S.T.J., *57*, No. 6, Part 2 (July-August 1978), pp. 1905-29.
3. B. W. Kerninghan and D. M. Ritchie, *The C Programming Language*, Englewood Cliffs, N. J. : Prentice-Hall, 1978.
4. B. R. Rowland and R. J. Welsch, "The 3B20D Processor & DMERT Operating System: Software Development System," B.S.T.J., this issue.